

Reconstruction of Aggregation Tree in spite of Faulty Nodes in Wireless Sensor Networks

Punit Sharma and Partha Sarathi Mandal

Indian Institute of Technology Guwahati

Guwahati - 781 039, India

Email: {psm, s.punit}@iitg.ernet.in

Abstract—Recent advances in wireless sensor networks (WSNs) have led to many new promising applications. However data communication between nodes consumes a large portion of the total energy of WSNs. Consequently efficient data aggregation technique can help greatly to reduce power consumption. Data aggregation has emerged as a basic approach in WSNs in order to reduce the number of transmissions of sensor nodes over *aggregation tree* and hence minimizing the overall power consumption in the network. If a sensor node fails during data aggregation then the aggregation tree is disconnected. Hence the WSNs rely on in-network aggregation for efficiency but a single faulty node can severely influence the outcome by contributing an arbitrary partial aggregate value.

In this paper we have presented a distributed algorithm that reconstruct the aggregation tree from the initial aggregation tree excluding the faulty sensor node. This is a synchronous model that is completed in several rounds. Our proposed scheme can handle multiple number of faulty nodes as well.

I. INTRODUCTION

A *wireless sensor networks* (WSNs) consist of a large number of spatially distributed autonomous resource-constrained tiny sensor devices which are used to lead many new promising applications. The applications for WSNs are varied, typically involving some kind of monitoring, tracking, or controlling. Specific applications include: Habitat monitoring, Object tracking, Nuclear reactor control, Fire detection, Traffic monitoring, etc. However data communication between nodes consumes a large portion of the total energy of WSNs. Consequently efficient data aggregation technique can help greatly to reduce power consumption. Data aggregation has emerged as a basic approach in WSNs in order to reduce the number of transmissions of sensor nodes over *aggregation tree* and hence minimizing the overall power consumption in the network.

Depending on the application, sensor nodes either report each and every measurement to a gateway or sink, or they perform in-network aggregation: En route to the sink, nodes combine their own measurement with the one of other nodes in proximity, e.g., their children on an *aggregation tree* rooted at the sink and spanning over all sensors [1]. A large fraction of WSNs requires only a periodic collection of an aggregate value (e.g., count, sum, average, etc.), and can do so with low network overhead. With in-network aggregation, rather than relaying individual measurements across multiple hops, each

node transmits a single packet, summarizing the data from an entire area of the WSNs.

Typically, there are three types of nodes in WSNs: leaf sensor nodes, aggregators, and a querier (sink) [2]. The aggregators collect data from a subset of the network, aggregate the data using a suitable aggregation function and then transmit the aggregated result to an upper aggregator or to the querier who generates the query. The querier is entrusted with the task of processing the received sensor data and derives meaningful information reflecting the events in the target field. It can be the base station or sometimes an external user who has permission to interact with the network depending on the network architecture. Data communications between sensors, aggregators and the queriers consume a large portion of the total energy consumption of the WSNs.

Most of the works [1]–[5] in literature focused on secure aggregation in WSNs. Secure aggregation means protecting data from attackers, where attackers intend to change the aggregation value and mislead the sink (or base station) resulting in false aggregation. They considered *faulty* node as an attacker or adversary that can compromise with sensor nodes by controlling their functionality and inducing arbitrary deviations from the protocols. But in our proposed algorithm, a faulty node is considered as a physical fault.

A sensor node is called *faulty*, if it cannot be able to communicate with any other sensor node in the WSNs. A sensor node may fail due to lack of battery power or some hardware failures. We may consider node failure as a permanent failure.

If a sensor node fails during data aggregation then the aggregation tree is disconnected. Hence the WSNs rely on in-network aggregation for efficiency but a single faulty node can severely influence the outcome by contributing an arbitrary partial aggregate value to the sink.

In a typical application, a WSN is scattered in a region where it is meant to collect data through its sensor nodes. We consider WSNs as a weighted communication graph, $G_c = (V, E)$ (say) where each sensor node is a vertex belonging to a set V and the communication link between two sensor nodes is defined as an edge belonging to a set E . Here edge weight is the cartesian distance between two sensor nodes. One node can communicate with other nodes directly if they are in its transmission range.

Using some distributed minimal spanning tree (MST) al-

gorithm [6] it is possible to construct an initial aggregation tree (T_a). If one node fails, then we assume, by some fault detection algorithm [3], that other nodes which are directly connected with the faulty node can detect the fault and the aggregation tree is decomposed into number of trees (disjoint-set of forest) with respect to the aggregation tree.

Our objective in the paper is as following: Given a weighted communication graph G_c and corresponding aggregation tree T_a with n nodes, if one arbitrary node, v_f (say) fails then how to reconstruct the aggregation tree with $n - 1$ nodes in a distributed way (excluding the faulty nodes), provided the reduced communication graph, $G'_c = (V', E')$ is still connected after removal of the faulty node, v_f where $V' = V \setminus \{v_f\}$ and $E' = E \setminus \{ \text{all edges are connected with } v_f \}$.

A. Related Work:

Chan *et al.* proposed a protocol [3] where they considered corrupted node as a malicious aggregator node. According to their protocol the answer given by aggregator is a good approximation of the true value even when the aggregator and a fraction of the sensor nodes are corrupted. In the paper [5] Haghani *et al.* considered adversary node as a misbehavior node that can severely influence the outcome by contributing an arbitrary partial aggregate value. Their scheme relies on costly operation to localize and exclude nodes that manipulate the aggregation when a fault is detected. Gallager *et al.* [6] proposed a distributed algorithm (distributive implementation of Prim's algorithm) constructing a MST of a connected graph in which the edge weights are unique. Their algorithm works on a message passing model. It uses a bottom-up approach and the overall message complexity of the MST algorithm is $O(E + n \lg n)$. In the paper [4] Gao and Zhu proposed a Dual-Head Cluster Based Secure Aggregation Scheme.

B. Our results:

The main contribution of this paper is a distributed algorithm for reconstruction of aggregation tree in wireless sensor networks when an arbitrary sensor node fail during aggregation. To the best of our knowledge, this is the first distributed protocol for reconstruction of aggregation tree which can handle multiple concurrent permanent node failure. Unlike Gallager *et al.* [6] algorithm the edge weights of underline communication graph may not be unique. We have proved that the reconstructed aggregation tree is again a MST. This is a synchronous model that completes in several rounds. In terms of rounds the complexity of our algorithm are $O(1)$ in the best case, $O(\lg n)$ in the worst case. The proposed algorithm can also handle multiple concurrent node failure.

II. RECONSTRUCTION OF AGGREGATION TREE

Consider the connected WSN consisting of n sensor nodes (vertices). Each sensor has its unique id, a variable (initially zero), where edge weight is the communication cost between two nodes. We assume that if one node fails the communication graph is still connected and by some fault detection algorithm neighbors of the faulty node can detect the fault.

We assume at a time there is only one faulty node in the WSN. Our proposed algorithm is synchronous; i.e., its perform in several rounds. Due to failure of a node, the aggregation decompose in to disjoint set of forest (cluster, say). According to the algorithm each cluster will find the minimum outgoing edge (synchronously) and tries to merge with the cluster on the other side of the edge. This is a distributed algorithm based on message passing.

A. Notations

Following notations are used throughout the paper for different type of message. These message are required during execution of the algorithm.

- *find_msg* (Find message): Fault detective node (cluster *root*, say) initiates the message within the cluster to invoke the node(s) for finding *moe*.
- *report_msg* (Report message): Every leaf node in the cluster sends a *report_msg* with *moe* information and own id to its parent after finding *moe* from it, and every intermediate node sends *report_msg* to its parent after getting information about the *moe* of its subtree including itself.
- *test_msg* (Test message): A node issue a *test_msg* message through the *moe* to know whether this edge is going to some other cluster.
- *accept_msg* (Accept message): A node generates a *accept_msg* message after receiving *test_msg* message if the *test_msg* message sender is belonging to different cluster.
- *reject_msg* (Reject message): A node generates a *reject_msg* message after receiving *test_msg* message if the *test_msg* message sender is belonging to the same cluster.
- *inform_msg* (Inform message): cluster *root* sends this message to the node in which the *moe* is attached.
- *merge_req* (Merge Request): Merging request from one cluster to some other cluster, containing *cluster id*.
- *internal_msg* (Internal message): This message is for pass the information in the same cluster.
- *merge_msg* (Merge message): To ensure merging between two cluster.
- *commit_msg* (Commit message): Final commitment
- *ignore_msg* (Ignore message): Ignore requests.
- *modify_msg* (Modify message): This message is generated by the end points of minimum outgoing edge after merging and pass in the new cluster to find the new *root*.

III. DESCRIPTION OF THE ALGORITHM

Suppose a sensor node with degree k is faulty in the initial aggregation tree T_a . Removal of this faulty node decomposes the aggregation tree into k number of trees (or clusters), T_1, T_2, \dots, T_k (say). Then let us assume by some fault detection algorithm the node, v_i^{df} (root of the cluster, say) directly attached with the faulty node in each cluster, T_i can find the information about the fault and starts following reconstruction process.

A. Subround-I: Minimum outgoing edge (moe) finding

For each cluster T_i , v_i^{df} named as *root* node initiates and sends *find_msg* to its descendents within the cluster through the tree edges with the *id* of the *root*, named as T_i^{id} , which is same as v_i^{df} . After receiving *find_msg* every other nodes assign T_i^{id} to its local variable (*cluster_id*) and forwards the message to neighbors until it reaches to leaf nodes. After receiving *find_msg* leaf node finds the *moe* and returns a *report_msg* to the sender of *find_msg*. After receiving *report_msg* all intermediate nodes modify *moe* if possible with respect to its own *moe* and forward the *report_msg* to the *root* node. For finding *moe* a node passes *test_msg* with *cluster_id* through the possible *moe* to test whether the other end of this *moe* is in the different cluster. If the other end of *moe* is in different cluster than the node returns a *accept_msg* with its own id otherwise the node returns a *reject_msg*.

After receiving *reject_msg* this node again tries to find the next possible *moe* among its neighbours until it receives a *accept_msg* or there is no possible *moe* edge for node. In that case the node marks all such rejected edges not to use further for *moe* selection. There may be a possibility of multiple *moe* at any individual node. In this case the node selects *moe* with minimum id node among the multiple *accept_msg*.

After receiving *report_msg* the *root* node finally selects a *moe* for the cluster and sends *inform_msg* to the corresponding node v_i^{moe} (say) attached with the *moe*.

B. Subround-II: Merge message passing

The node, v_i^{moe} of each cluster, T_i sends a *merge_req* message along their respective *moe* to some node of T_j , say. The decision after receiving *merge_req* message as following: There are two cases:

- 1) If *root*, v_j^{df} of T_j receives *merge_req* and if the *cluster_id* of T_j is less than the *cluster_id* of T_i then v_j^{df} returns an *ignore_msg* to v_i^{moe} , otherwise v_j^{df} keeps the information in its database.
- 2) If some other node (v_j) excluding v_j^{df} of T_j receives a *merge_req* and if *cluster_id* of T_j is less than the *cluster_id* of T_i then the node v_j returns an *ignore_msg* to v_i^{moe} , otherwise v_j forwards the message (*internal_msg*) to the *root* v_j^{df} .

C. Subround-III: Decision after receiving a merge messages

At the end of the previous Subround-II if *root* of T_j for some j receives one or more than one *merge_req* messages then it finds the minimum *cluster_id* over all messages and sends a *merge_msg* to the minimum id cluster and sends *ignore_msg* to all others directly or via v_j node (v_j is considered in the case-2 of Subround-II). Now, if *root* of T_j for some j does not receive any *merge_req* or receive but pass a *ignore_msg* to sender then the *root* of T_j sends a *merge_msg* through the *moe* (chosen in Subround-II) from v_j^{moe} node.

D. Subround-IV: Merging of clusters

In this subround each cluster T_i , for $i = 1, 2, \dots, k$ some node v_i (including *root*) receives *merge_msg* and/or *ignore_msg* from v_j (including *root*) of some other cluster T_j . If the message is *ignore_msg* then drop the message. Otherwise merge these two clusters in the following ways:

- 1) If v_i sends a *merge_msg* to v_j and if $T_i^{id} < T_j^{id}$ then T_i sends a *commit_msg* to T_j and T_j merge with T_i by including the edge in the modified aggregation tree. After that the vertices attached with the edge initiate *modify_msg* over the new cluster T'_i (, say) with the information of v_i^{df} for the modification of *root*. If v_i sends a *merge_msg* to v_j and if $T_i^{id} > T_j^{id}$ then *merge_msg* is drop without merging.
- 2) If v_i does not send a *merge_msg* to T_j then v_i sends a *commit_msg* and a *modify_msg* (as a responds) to cluster T_j after receiving *modify_msg* from its own cluster. Then T_j merge with T_i by including the edge in the modified aggregation tree and T_j expand.

IV. THE ALGORITHM

```

 $G_c = (V, E) \leftarrow$  Communication graph
 $T_a \leftarrow$  Initial aggregation tree
 $k \leftarrow$  Degree of the faulty node,  $v_f$ 
Subround-I: (Finding moe)
for each cluster  $T_i$  ;  $i = 1$  to  $k$  do
     $root$ ,  $v_i^{df}$  initiates and sends  $< find\_msg, T_i^{id} >$ 
    for each node  $v_i$  do
         $cluster\_id_i \leftarrow T_i^{id}$ 
    end for
    for each node  $v_i$  (starts from leaf nodes) do
        passes test_msg through its moe  $\in E'$  of  $G'_c$  to some
        other node  $v_{i'}$ 
        if  $T_i^{id} \neq T_{i'}^{id}$  then
             $v_{i'}$  returns an accept_msg to  $v_i$ 
             $v_i$  passes a reprot_msg to the find_msg sender
        else
             $v_{i'}$  returns reject_msg to  $v_i$  and marks this rejected
            edge in  $E'$  and  $v_i$  looks for the next possible moe
        end if
    end for
    for each node  $v_i$  (intermediate/root) do
        After receiving report_msg the node modifies moe
        if possible wrt its own moe as above and forwards
        report_msg to its ancestor until it reaches to the root
    When root receives the  $v_i^{moe}$  then it passes the
    inform_msg to the  $v_i^{moe}$  if moe is not attached with
    the root
end for
end for
if there is no moe then
    return Tree is reconstructed & the protocol is terminated
else
    moves for the Subround-II

```

```

end if
Subround-II: (Merge message passing)
for each cluster  $T_i$  ;  $i = 1$  to  $k$  do
     $v_i^{moe}$  sends a merge_req from cluster  $T_i$  to some  $T_j$ 
end for
if  $v_j^{df}$  of  $T_j$  receives this merge_req message then
    if  $C_j^{id} < C_i^{id}$  then
        passes an ignore_msg to  $v_i^{moe}$ 
    else
        keeps the message
    end if
else
    if some other node  $v_j$  of  $T_j$  receives this merge_req message then
        if  $C_j^{id} < C_i^{id}$  then
            passes an ignore_msg to  $v_i^{moe}$ 
        else
             $v_j$  receives this message and passes it to  $v_j^{df}$  of  $T_j$ 
            through an internal_msg
        end if
    end if
end if
Subround-III: (Decision after receiving a merge messages)
for each cluster  $T_i$  ;  $i = 1$  to  $k$  do
    if  $v_i^{df}$  receives merge_req from some other clusters then
        sends merge_msg to the minimum id cluster among
        them and ignore_msg to others
    else
        sends merge_msg from  $v_i^{moe}$  through moe
    end if
end for
Subround-IV: (Merging of clusters)
 $v^i$  of  $T_i$  receives either merge_msg or/and ignore_msg
from  $T_j$  after the end of Subround-III
if message is ignore_msg then
    drops the message without merging
else
    if  $T_i$  also sends a merge_msg to  $T_j$  then
        if  $T_i^{id} < T_j^{id}$  then
             $T_i$  passes a commit_msg to  $T_j$ 
             $T_j$  merges with  $T_i$  in some new cluster  $T'_i$ 
            the nodes attached with merged edge initiates and
            sends modify_msg within  $T'_i$ .
             $cluster\_id$  of  $v_{i'} \in T'_i$  resets the value by  $v_i^{df}$ 
        else
            drops this received merge_msg
        end if
    else
         $v^i$  sends a commit_msg and forwards modify_msg
        (as a responds) to cluster  $T_j$  after receiving
        modify_msg from its own cluster and then  $T_j$  merges
        with  $T_i$ 
    end if
end if

```

Re-execute the protocol from Subround-I with modified clusters until termination.

V. COMPLEXITY ANALYSIS

Let k be the number of clusters after a node failure. We are measuring the complexity of the proposed algorithm in terms of rounds of execution and total number of message exchange. First we concentrate over possible best and worst rounds of execution.

- Case-1 (Best Case) If v_i^{moe} sends *merge_req* to the minimum id cluster T_j (say) for all $i \in \{1, 2, \dots, k\} \setminus \{j\}$, then the tree would be reconstructed in one round.
- Case-2 (Worst Case) If every distinct pair of clusters exchange *merge_req* in Subround-II and merge in Subround-IV then in one round number of cluster reduces by half. If this kind of merging process is continue then after $O(\lg k)$ rounds the tree would be reconstructed.

Now we determine an upper bound for the number of messages for a cluster T_i .

Let the number of nodes in this cluster is n_i . Recall the types of messages used by the algorithm :

find_msg: $n_i - 1$ *find_msg* messages.

test_msg: (successful test and failed test.)

accept_msg: Acceptance requires two messages, successful test and accept. So the messages are $2n_i$. Note that *inform_msg* also included in this count.

reject_msg: Note that an edge can be reject at most once throughout the execution of the algorithm. Rejection requires two messages: failed test and reject. So we have $2E$ messages.

report_msg: $n_i - 1$ *report_msg*.

merge_req: 1 (one) request for merging.

ignore_msg: at most $k - 1$ *ignore_msg* throughout the execution of the algorithm.

internal_msg: at most $n_i - 1$ message.

merge_msg: one message.

commit_msg: one message for final commitment

modify_msg: $n_i - 1$ messages for modification.

The total number of message required for a cluster is $6n_i$. Total number of message for all k clusters is $\sum_{i=1}^k (6n_i) = 6(n - 1)$ where $n - 1 = \sum_{i=1}^k n_i$

Therefore the total number of message for merging of all k clusters is $O(n \lg k + E)$. Here k may be $n - 1$, therefore the total counting brings us to $O(n \lg n + E)$.

VI. CORRECTNESS

Note that in a single round of proposed algorithm, every cluster sends a unique *merge_msg* through *moe*. In the merging of two or more than two clusters simultaneously there is exactly two clusters which sends a *merge_msg* to each other through the same *moe*.

Theorem 1: There is no cycle after merging two or more clusters.

Proof: Let T_a be the initial aggregation tree with n nodes and v_f be the faulty node. Proof by induction on degree of v_f node in T_a .

Basis: Let $\deg(v_f) = 1$. Then after removing v_f from T_a , there is only one cluster with $n - 1$ nodes. Clearly T'_a with $n - 1$ nodes is again a tree.

Let $\deg(v_f) = 2$ and T_i, T_j be the clusters. Let us suppose cycle occurs in the merging of T_i and T_j . It is possible if both T_i and T_j send a *merge_msg* to each other through different multiple *moe*. But this contradicts Subround-III of the proposed algorithm. Since according to proposed algorithm both T_i and T_j send a *merge_msg* to each other through same *moe*. Hence there is no cycle in the merging of T_i and T_j .

Inductive hypothesis: Let no cycle occurs in the merging of k or less clusters, i.e., $\deg(v_f) \leq k$.

Inductive step: Now let $\deg(v_f) = k + 1$ and T_i , for $i = 1, 2, \dots, k + 1$ be the clusters. Let us suppose cycle occurs in the merging of these $k + 1$ clusters. It is possible if at least three cluster T_1, T_2, T_3 (, say) send the *merge_msg* to each other as T_1 to T_2 , T_2 to T_3 , T_3 to T_1 in a round. But this contradicts our algorithm that there are exactly two clusters which send a *merge_msg* to each other through the same *moe* in the merging of more than two clusters. Therefore cycle cannot occur in a round and number of clusters reduces. Now by inductive hypothesis cycle will not occur in the merging of $k + 1$ clusters. Hence theorem is true for any number of clusters. ■

Theorem 2: Resultant reconstructed aggregation tree is again a MST.

Proof: Let T_a be the initial aggregation tree and given that is a MST with n nodes and v_f be the faulty node with degree k . Let T'_a be the aggregation tree which is reconstructed using our proposed algorithm with $n - 1$ nodes after removing the faulty node v_f . Since T_a is a MST, therefore removal of v_f divides it in to k sub trees where each of them are individually a MST. Now suppose T'_a is not a MST, it means there are at least two clusters which is not merged with a minimum weighted edge in the T'_a . But it is a contradiction of our algorithm that allows merging between different clusters through a minimal weighted edge. Hence the resultant reconstructed aggregation tree is again a MST. ■

VII. MULTIPLE SENSOR NODES FAILURE

If m number of nodes fail simultaneously and if d_1, d_2, \dots, d_m are the degrees of respective faulty nodes then at most $d_1 + d_2 + \dots + d_m$ number of disjoint forest may form. Then same proposed algorithm can merge all disjoint forest and reconstruct the aggregation tree.

VIII. CONCLUSION

In this paper, we have proposed a distributed algorithm for reconstruction of aggregation tree in wireless sensor networks when an arbitrary sensor node fails during aggregation. Our model is synchronous, performing in rounds. In terms of rounds the time complexity of our algorithm is $O(1)$ in the best case, $O(\lg n)$ in the worst case. Our proposed algorithm can also handle multiple concurrent sensor node failure. But the proposed algorithm cannot handle node failure during the reconstruction phase. In our future works we will try to

incorporate node failure during the reconstruction phase as well.

REFERENCES

- [1] F. Y.-S. Lin, H.-H. Yen, and S.-P. Lin, "A novel energy-efficient mac aware data aggregation routing in wireless sensor networks," *Sensors*, vol. 9, no. 3, pp. 1518–1533, 2009.
- [2] H. Alzaid, E. Foo, and J. G. Nieto, "Secure data aggregation in wireless sensor network: a survey," in *AISC '08: Proceedings of the sixth Australasian conference on Information security*. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2008, pp. 93–105.
- [3] H. Chan, A. Perrig, B. Przydatek, and D. X. Song, "Sia: Secure information aggregation in sensor networks," *Journal of Computer Security*, vol. 15, no. 1, pp. 69–102, 2007.
- [4] F. Gao and W. T. Zhu, "A dual-head cluster based secure aggregation scheme for sensor networks," *Network and Parallel Computing Workshops, IFIP International Conference on*, vol. 0, pp. 103–110, 2008.
- [5] P. Haghani, P. Papadimitratos, M. Poturalski, K. Aberer, and J.-P. Hubaux, "Efficient and robust secure aggregation for sensor networks," in *NPSEC '07: Proceedings of the 3rd IEEE Workshop on Secure Network Protocols*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 1–6.
- [6] R. G. Gallager, P. A. Humblet, and P. M. Spira, "A distributed algorithm for minimum-weight spanning trees," *ACM Trans. Program. Lang. Syst.*, vol. 5, no. 1, pp. 66–77, 1983.